

IGL502/752 – Techniques de vérification et de validation
Université de Sherbrooke

Examen final

Enseignant: Michael Blondin
Date: jeudi 14 décembre 2023
Durée: 3 heures

Directives:

- Vous devez répondre aux questions dans le **cahier de réponses**, et non sur ce questionnaire;
- **Une seule feuille** de notes au format $8\frac{1}{2}'' \times 11''$ est permise;
- Les **fiches récapitulatives** se trouvent à la fin du questionnaire;
- **Aucun matériel additionnel** (notes de cours, fiches récapitulatives, etc.) n'est permis;
- **Aucun appareil électronique** (calculatrice, téléphone, montre intelligente, etc.) n'est permis;
- Vous devez donner **une seule réponse** par sous-question;
- L'examen comporte **6 questions** sur **3 pages** valant un total de **50 points**;
- La correction se base notamment sur la **clarté**, l'**exactitude** et la **concision** de vos réponses, ainsi que sur la **justification** pour les questions qui en requièrent une.

Question 1: logique temporelle linéaire (LTL)

(a)

6 pts

(b)

2 pts

(c)

3 pts

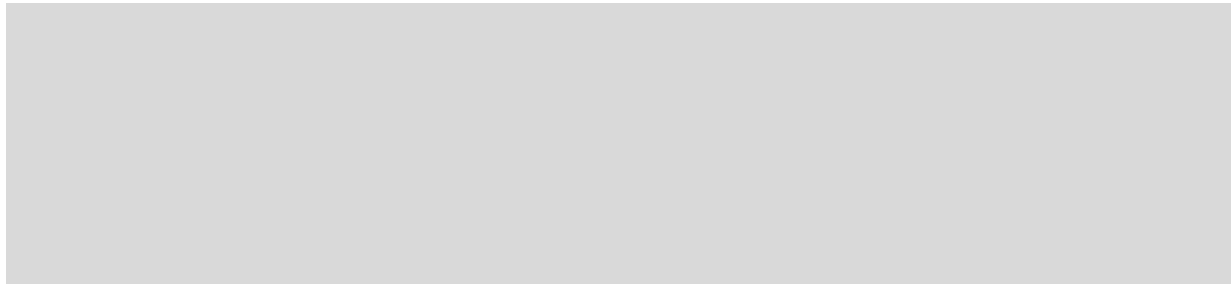
Question 2: automates de Büchi

(a)

4 pts

(b)

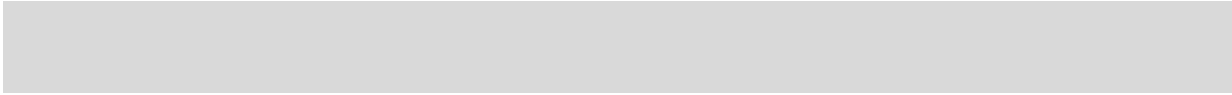
2 pts

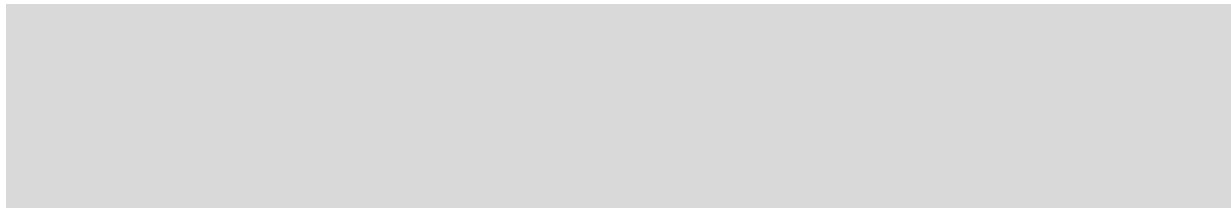
Question 3: logique temporelle arborescente (CTL) et vérification symbolique

(a)  6 pts

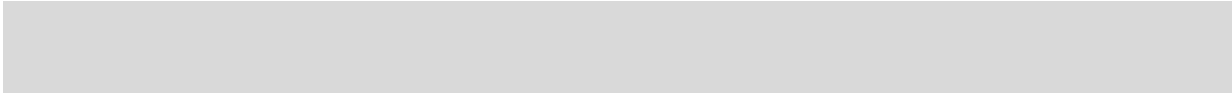
(b)  2 pts

(c)  2 pts

(d)  2 pts

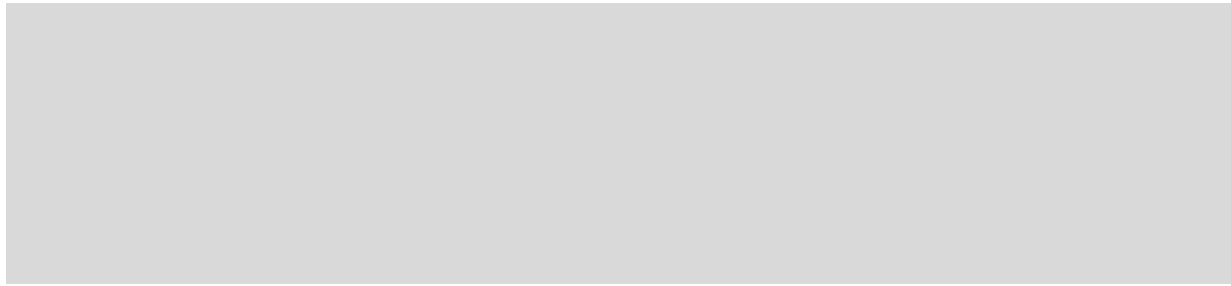
Question 4: systèmes à pile

(a)  2,5 pts

(b)  1 pt

(c)  2,5 pts

(d)  1 pt

Question 5: systèmes à compteurs

(a)



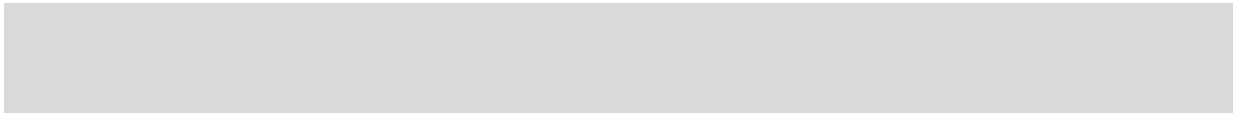
3 pts

(b)

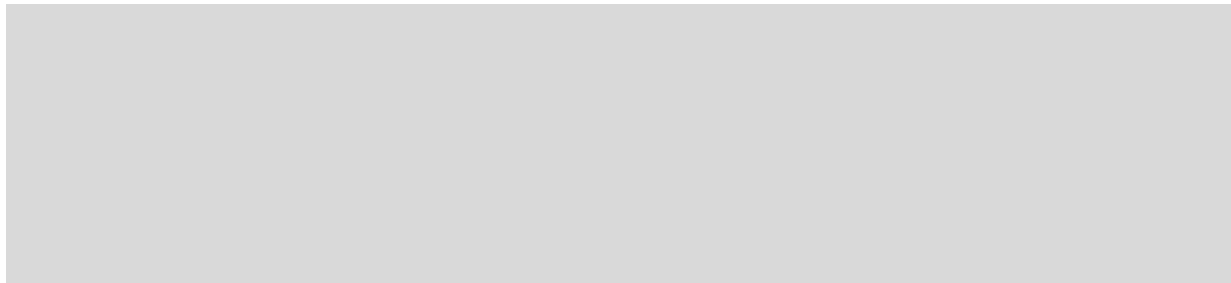


1,5 pts

(c)



2,5 pts

Question 6: systèmes probabilistes

(a)



2 pts

(b)



2 pts

(c)



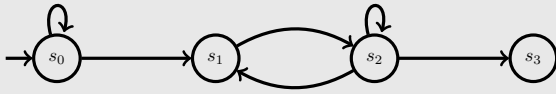
3 pts

FICHES RÉCAPITULATIVES

1. Systèmes de transition

Systèmes de transition

- Modélise formellement un système concret
- États: ensemble S (sommets)
- Relation de transition: $\rightarrow \subseteq S \times S$ (arcs)
- États initiaux: $I \subseteq S$ (où S peut débuter)



Prédécesseurs et successeurs

- Successeurs immédiats: $\text{Post}(s_1) = \{s_2\}$
- Prédécesseurs immédiats: $\text{Pre}(s_1) = \{s_0, s_2\}$
- Successeurs: $\text{Post}^*(s_1) = \{s_1, s_2, s_3\}$
- Prédécesseurs: $\text{Pre}^*(s_1) = \{s_0, s_1, s_2\}$
- États terminaux: s_3 car $\text{Post}(s_3) = \emptyset$

Chemins et exécutions

- Chemin fini: $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_1$
- Chemin infini: $s_1 \rightarrow s_2 \rightarrow s_1 \rightarrow s_2 \rightarrow \dots$
- Ch. maximal: ne peut être étendu, par ex. $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3$
- Exécution: chemin maximal qui débute par un état initial

Structures de Kripke

- Décrit les propriétés des états d'un système
- Système de transition $(S, \rightarrow, I)$
- Propositions atomiques AP
- Fonction d'étiquetage $L: S \rightarrow 2^{AP}$
- Exemple: si $AP = \{p, q, r\}$ et $L(s_1) = \{p, q\}$, alors s_1 satisfait p et q , mais pas r

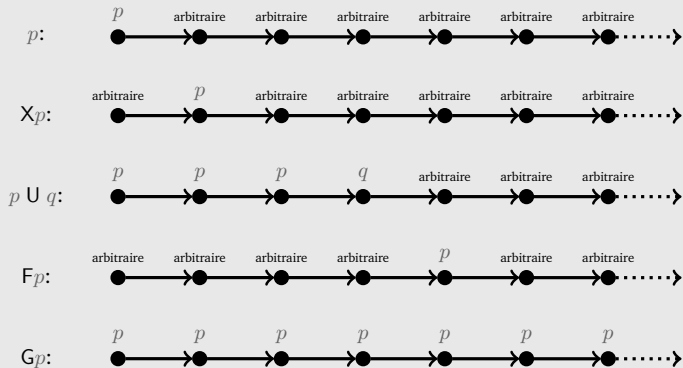
Explosion combinatoire

- Nombre d'états peut croître rapidement, par ex. $\mathcal{O}(2^n)$
- Existe techniques pour surmonter ce problème

2. Logique temporelle linéaire (LTL)

Logique

- Intérêt: spécifier formellement des propriétés
- Syntaxe: $\text{vrai} \mid p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg \varphi \mid X\varphi \mid \varphi \cup \varphi \mid F\varphi \mid G\varphi$
- Interprétation: sur des traces, c.-à-d. mots infinis de $(2^{AP})^\omega$
- Sémantique:



Équivalences

- Distributivité: X, G, U (gauche) sur $\wedge$; X, F, U (droite) sur $\vee$
- Dualité: X dual de lui-même, F dual de G
- Autre: seules combinaisons de F et G : $\{F, G, FG, GF\}$

Types de propriétés

- Invariant: propriété toujours vraie: $G\varphi$
- Sûreté: réfutable avec préfixe fini
- Vivacité: comportements vers l'infini

Vérification

- Trace: états d'une exécution infinie $\mapsto$ leurs étiquettes
- Satisfaisabilité: $\mathcal{T} \models \varphi \iff \text{Traces}(\mathcal{T}) \subseteq \llbracket \varphi \rrbracket$
- Équité: omettre traces triviales où un processus est ignoré
- En pratique: Spin (avec Promela), par ex. algorithme de Lamport, protocole de Needham-Schroeder

3. Langages ω -réguliers

Expressions ω -régulières

- Décrivent: les langages ω -réguliers de mots infinis
- Syntaxe:

$$s ::= r^\omega \mid (r \cdot s) \mid (s + s)$$

$$r ::= r^* \mid (r \cdot r) \mid (r + r) \mid a \mid \varepsilon$$

- Exemples:

$a(a+b)^\omega$: mots qui débutent par a ,

$(ab)^\omega$: l'unique mot $abababab \dots$

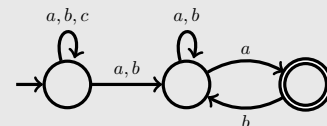
$b^*(aa^*bb^*)^\omega$: mots avec une infinité de a et de b

$(a+b)^*b^\omega$: mots avec un nombre fini de a

Automates de Büchi

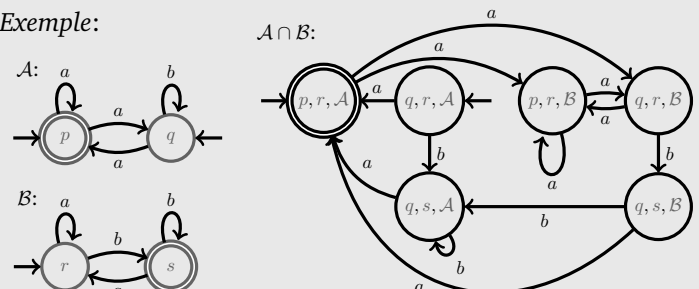
- Définition: automates usuels; plusieurs états initiaux
- Langage: mots qui visitent états acceptants ∞ souvent
- Expressivité: $\equiv$ expressions ω -rég.; déterminisme $\neq$ non dét.

- Exemple: mots tels que $\#a = \infty$, $\#b = \infty$ et $\#c \neq \infty$:



Intersection d'automates de Büchi

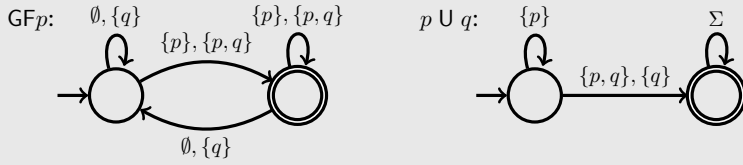
- 1^{ère} idée: simuler $\mathcal{A}$ et $\mathcal{B}$ en parallèle via produit; pas suffisant
- Solution: faire deux copies, alterner aux états acceptants
- Exemple:



4. Vérification algorithmique de formules LTL

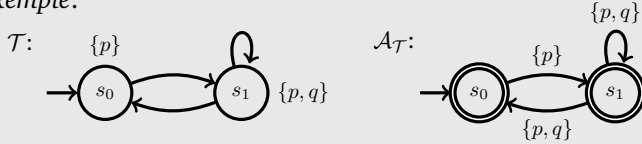
LTL vers automates

- **Alphabet:** $\Sigma := 2^{AP}$
- **Conversion:** $\varphi \rightarrow \mathcal{A}_\varphi$ (pire cas: $2^{\mathcal{O}(|\varphi|)}$ états)
- **Exemples:**



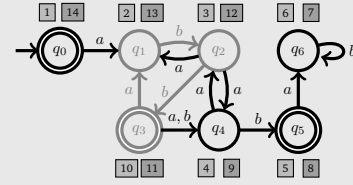
Structures de Kripke vers automates

- **Conversion:** étiquettes $\equiv$ lettres + tout acceptant
- **Exemple:**

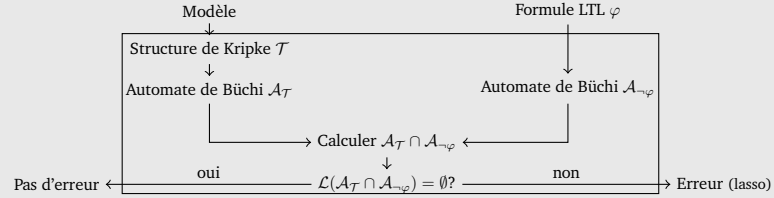


Test du vide

- **Vérification:** $\mathcal{T} \models \varphi \iff \mathcal{L}(\mathcal{A}_\mathcal{T}) \cap \mathcal{L}(\mathcal{A}_{\neg\varphi}) = \emptyset$
- **Lassos:** $\mathcal{L}(\mathcal{B}) \neq \emptyset \iff \exists q_0 \xrightarrow{*} q \xrightarrow{+} q$ où $q_0 \in Q_0, q \in F$
- **Détection:** double parcours en profondeur (temps linéaire)



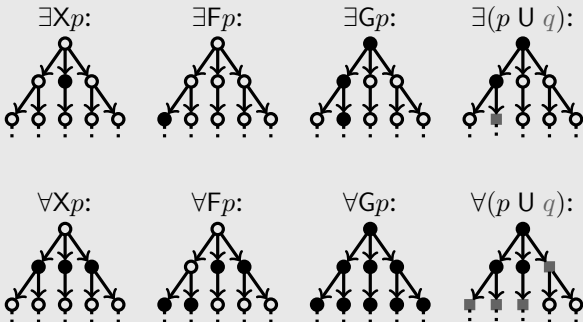
- **Mieux:** identifier les composantes fort. conn. (temps linéaire)
- **Sommaire:**



5. Logique temporelle arborescente (CTL)

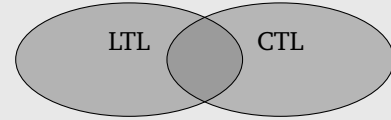
Logique

- **Intérêt:** raisonne sur le temps avec un futur indéterminé
- **Syntaxe:** $\text{vrai} \mid p \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \neg\Phi \mid QT\Phi \mid Q(\Phi \cup \Phi)$
où $Q \in \{\exists, \forall\}, T \in \{X, F, G\}$
- **Interprétation:** sur l'arbre de calcul d'une structure de Kripke
- **Sémantique:**



Propriétés d'un système

- **Satisfiabilité dépend des états:** $\llbracket \Phi \rrbracket := \{s \in S : s \models \Phi\}$
- **Spécification:** $\mathcal{T} \models \Phi \iff I \subseteq \llbracket \Phi \rrbracket$
- **Expressivité:** incomparable à LTL



Équivalences

- **Distributivité:**

$$\begin{aligned} \exists F(\Phi_1 \vee \Phi_2) &\equiv (\exists F\Phi_1) \vee (\exists F\Phi_2) \\ \forall G(\Phi_1 \wedge \Phi_2) &\equiv (\forall G\Phi_1) \wedge (\forall G\Phi_2) \end{aligned}$$

- **Attention:** pas équiv. si on change les quantificateurs
- **Dualité:** effet d'une négation: $\exists \leftrightarrow \forall, X \leftrightarrow X \text{ et } F \leftrightarrow G$
- **Idempotence:** $QTQT\Phi \equiv QT\Phi$ où $Q \in \{\exists, \forall\}, T \in \{F, G\}$

6. Vérification algorithmique de formules CTL

Algorithme

- **Approche:** calculer $\llbracket \Phi' \rrbracket$ pour chaque sous-formule Φ' de Φ
- **Vérification:** tester $I \subseteq \llbracket \Phi \rrbracket$
- **Forme normale existentielle** plus simple, mais pas nécessaire
- **Complexité:** $\mathcal{O}((|S| + |\rightarrow|) \cdot |\Phi|)$ avec bonne implémentation
- **En pratique:** NuSMV + langage de description de haut niveau

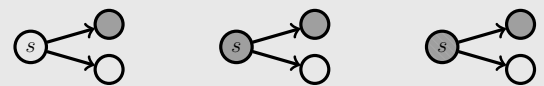
Logique propositionnelle

- **Règles récursives:**

$$\begin{aligned} \llbracket \text{vrai} \rrbracket &= S, \\ \llbracket p \rrbracket &= \{s \in S : p \in L(s)\}, \\ \llbracket \Phi_1 \wedge \Phi_2 \rrbracket &= \llbracket \Phi_1 \rrbracket \cap \llbracket \Phi_2 \rrbracket, \\ \llbracket \neg\Phi \rrbracket &= S \setminus \llbracket \Phi \rrbracket. \end{aligned}$$

Opérateurs temporels existentiels

- **Calcul de $\llbracket \exists X\Phi \rrbracket$:** $\{s \in S : \text{Post}(s) \cap \llbracket \Phi \rrbracket \neq \emptyset\}$
- **Calcul de $\llbracket \exists G\Phi \rrbracket$:** $T \subseteq \llbracket \Phi \rrbracket$ max. t.q. $\forall s \in T : \text{Post}(s) \cap T \neq \emptyset$
- **Calcul de $\exists(\Phi_1 \cup \Phi_2)$:** $T \supseteq \llbracket \Phi_2 \rrbracket$ min. t.q.
 $s \in \llbracket \Phi_1 \rrbracket \wedge \text{Post}(s) \cap T \neq \emptyset \implies s \in T$



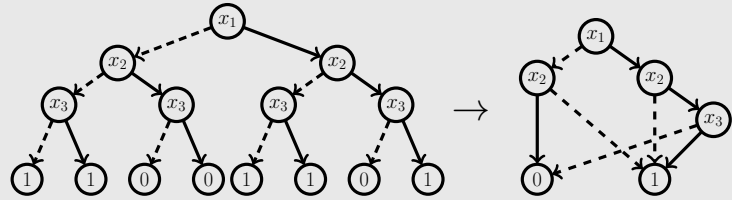
Optimisations

- **Autres opérateurs:** $\forall$ et F s'implémentent directement; nécessaire pour obtenir un algorithme polynomial
- **Points fixes:** temps linéaire si calculs directs sans raffinements itératifs; par ex. calcul de composantes fortement connexes

7. Vérification symbolique : diagrammes de décision binaire

Diagramme de décision binaire

- **But:** représenter des fonctions booléennes de façon compacte
- **Utilité:** manipuler efficacement de grands ensembles d'états
- **Propriétés:**
 - graphe dirigé acyclique
 - sommets étiquetés par variables ordonnées sauf 0 et 1
 - les chemins respectent l'ordre des variables
 - sommets *uniques* et *non redondants* ($lo(u) \neq hi(u)$)
- **Canonicité:** pas deux BDDs pour la même fonction booléenne

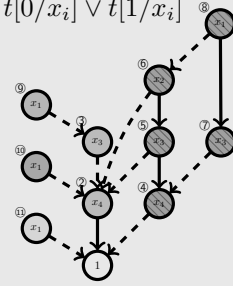


Manipulation

- **Représentation:** tableau associatif $sommet \leftrightarrow (x_i, lo, hi)$
- **Ajout d'un sommet:** temps constant avec $make(x_i, lo, hi)$
- **Construction:** par substitutions récursives avec $build(\varphi)$
- **Op. bool.:** application récursive « synchronisée » avec $apply_{\circ}$
- **Quantif.:** $\exists x_i \in \{0, 1\} : t$ obtenu via $t[0/x_i] \vee t[1/x_i]$
- **Complexité:** polynomiale sauf pour $build$

Vérification

- **État:** représenté par identifiant binaire
- **Transition:** paire d'identifiants binaires
- **Ensemble:** représenté par sommet de BDD
- **Logique prop.:** manipulation de BDD
- **Opérateurs temporels:** via calculs de Post ou Pre sur BDD
- **Satisfiabilité:** $I \subseteq \llbracket \Phi \rrbracket \Leftrightarrow I \cap \llbracket \Phi \rrbracket = \emptyset \Leftrightarrow apply_{\wedge}(u_I, u_{\llbracket \Phi \rrbracket}) = 0$



8. Systèmes avec récursion

Contexte

- **Espace d'états:** pile d'appel ou d'éléments (+ valeurs locales)
- **Défi:** gérer un nombre infini ou arbitraire d'états
- **Approche:** construction et analyse de systèmes à pile

Systèmes à pile

- **Définition:** états P , alphabet Γ , transitions $\{p \xrightarrow{a \rightarrow u} q, \dots\}$
- **But:** décrire un ensemble de piles (et non accepter des mots)
- **Configuration:** $\langle p, w \rangle \in P \times \Gamma^* \mapsto \text{pile } \begin{matrix} w_1 \\ \vdots \\ w_n \end{matrix}$
- **Exemple de modélisation:**

bool $x \in \{\text{faux}, \text{vrai}\}$

foo():

$x = \neg x$;

si x : foo();

sinon: bar();

retourner

bar():

si x : foo();

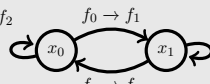
retourner

$f_1 \rightarrow b_0 f_2$

$f_2 \rightarrow \varepsilon$

$b_0 \rightarrow b_1$

$b_1 \rightarrow \varepsilon$



$f_1 \rightarrow f_0 f_2$

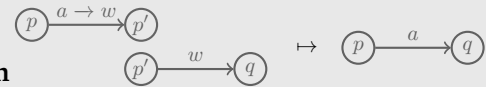
$f_2 \rightarrow \varepsilon$

$b_0 \rightarrow f_0 b_1$

$b_1 \rightarrow \varepsilon$

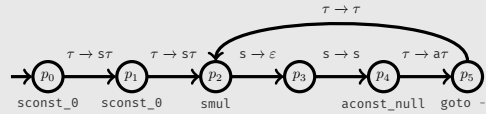
Calcul de prédécesseurs/successeurs

- **Déf.:** $Pre^*(C) := \bigcup_{i \geq 0} Pre^i(C)$ et $Post^*(C) := \bigcup_{i \geq 0} Post^i(C)$
- **Représentation:** symbolique de C avec un $\mathcal{P}$ -automate $\mathcal{A}$
- **Idée:** (états initiaux = états de $\mathcal{P}$) + mots sur alphabet de pile
- **Décrit:** $Conf(\mathcal{A}) := \{\langle p, w \rangle : p \xrightarrow{w}_{\mathcal{A}} \odot\}$
- **Algorithme:** permet de calculer $Pre^*(Conf(\mathcal{A}))$ par saturation
- **Approche:** init. $\mathcal{B} := \mathcal{A}$ puis enrichir avec cette règle:



Vérification

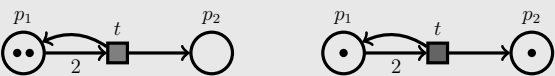
- **Approche:** système $\mapsto$ sys. à pile, spécification $\mapsto \mathcal{P}$ -automate, vérification: par $Pre^*/Post^*/$ automate de Büchi (LTL)
- **Applications:** raisonnement sur piles, par ex. « bytecode »



9. Systèmes infinis

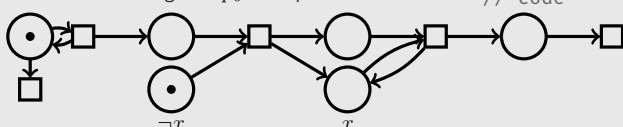
Réseaux de Petri

- **Déf.:** places et transitions reliées par arcs pondérés
- **Marquage:** nombre de jetons par place
- **Déclenchement:** si assez de jetons pour chaque arc entrant, les retirer, et en ajouter de nouveaux selon les arcs sortants
- **Successeurs:** $Post^*(m) = \{m' \in \mathbb{N}^P : m \xrightarrow{*} m'\}$
- **Prédécesseurs:** $Pre^*(m') = \{m \in \mathbb{N}^P : m \xrightarrow{*} m'\}$
- **Exemple:**



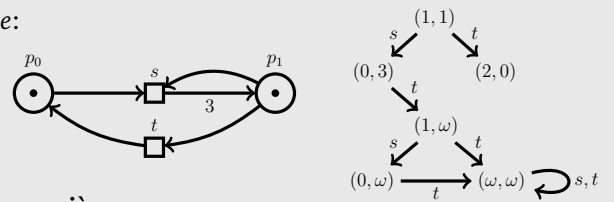
Modélisation

- **Processus:** comptés par les places
- **Exemple:** si $\neg x$: $x = \text{vrai}$ tant que $\neg x$:
sinon: goto p_0 pass // code



Graphes de couverture

- **Idée:** construire $Post^*(m)$ mais accélérer avec ω si $x < x'$
- **Test:** m' couvrable ssi le graphe contient un $m'' \geq m'$
- **Exemple:**

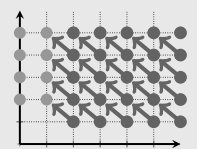


Algorithme arrière

- **Idée:** construire $\uparrow Pre^*(\uparrow m')$ en déclenchant vers l'arrière
- **Représentation:** d'ensemble clos par le haut par base finie
- **Test:** m peut couvrir m' ssi découvert

Accessibilité

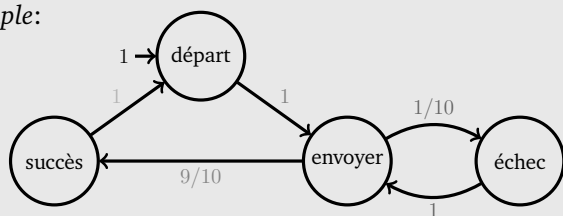
- **Problème:** tester si $m' \in Post^*(m)$
- **Décidable** mais plus compliqué



10. Systèmes probabilistes

Chaîne de Markov

- *But*: remplacer non-déterminisme par probabilités
- *Déf.*: struct. de Kripke avec proba. sur transitions / états init.
- *Représentation*: probabilités = matrice $\mathbf{P}$ et vecteur **init**
- *Exemple*:



- *Événements*: exéc. inf. décrites par préfixes finis (cylindres)
- *Probabilité*: somme du produit des transitions de cylindres
- *Exemple*: $\mathbb{P}(\text{F succès}) = \sum_{i=0}^{\infty} 1 \cdot ((1/10) \cdot 1)^i \cdot (9/10) \cdot 1 = 1$
- *Outils*: PRISM/Storm (PCTL, analyse quantitative, etc.)

Accessibilité

- *Accessibilité*: événement de la forme $A \cup B$
- *Partition*: $S_0 := \llbracket \neg \exists (A \cup B) \rrbracket$, $S_1 := B$, $S_2 := S \setminus (S_0 \cup S_1)$
prob. nulle prob. = 1 prob. à déterminer
- *Approche*: $\mathbf{A} := \mathbf{P}$ sur S_2 ; $\mathbf{b}(s) :=$ proba. d'aller de s vers S_1 ; $\mathbf{x}(s) = \mathbb{P}(s \models A \cup B)$ est la solution de $(\mathbf{I} - \mathbf{A}) \cdot \mathbf{x} = \mathbf{b}$
- *Approx.*: $A \cup^{\leq n} B$ obtenu par $f^n(\mathbf{0})$ où $f(\mathbf{y}) := \mathbf{A} \cdot \mathbf{y} + \mathbf{b}$

Comportements limites

- *CFC terminales*: une est atteinte et parcourue avec proba. 1
- *FG et GF*: se calculent via accessibilité et CFC terminales

CTL probabiliste (PCTL)

- *Syntaxe*: comme CTL, mais $\exists / \forall$ deviennent $\mathcal{P}_I$, et ajout $\cup^{\leq n}$
- $\mathcal{P}_I(\varphi)$: proba. de φ dans intervalle I ?
- $\cup^{\leq n}$: côté droit satisfait en $\leq n$ étapes?
- *Vérification*: calcul récursif + éval. proba. d'accessibilité